[430-555] Homework 4-1 : MVCC Design for Column-Store

Student Name: LE VAN DUC **Student ID: 2016-27885**

1. Data space and Version space:

- **Data space** keeps most committed data. In Column-Store, we have a Column structure which keeps encoded value from a Dictionary encoding structure. The row id (rid) of Column is implied by its index. Each row in Column structure has a Commit Sequence Number (CSN) which is the Commit Time stamp of the Transaction that updates this row. The row has 1 field named version_flag to indicate this row has versions (version_flag = 1) or not (version_flag = 0).

Data space - Column :

Column’s encoded value			
rid (implicit)	**version_flag**	**encoded value**	**CSN**
0	0	10	250
1	1	20	300
2	0	30	350
...			

Column’s Dictionary	
index (implicit)	**dictionary value**
10	“Hello”
20	“How are you”
30	“Korea”

Data structure:

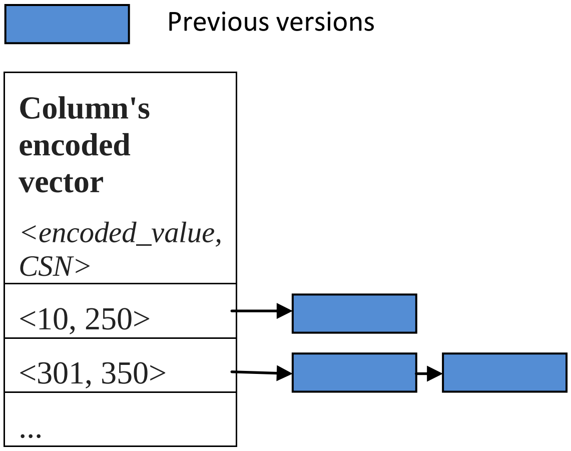
```
struct column_value {

            boolean version_flag;

            unsigned long encoded_value;

            unsigned long csn;

}

vector<column_value> column; // column data structure

vector<string> dictionary;      // dictionary data structure
```

- **Version space** keeps all history versions of Data space. Similar to Data space, in Version space, each Column will have a Version data structure to keep version encoded data and an encoding Dictionary.

To **reduce the frequently update** on Dictionary then we will use a **Delta space** which keeps the **values new** on Column’s Dictionary. The index of Delta space will **start from max index** on Column’s Dictionary (to not change Column encoded values).

Each version will have a CSN number equals to Commit Time stamp of the Transaction creating this version and has a pointer to the previous version so that we can travel through all versions.

Version space – Column :



🟦	Previous versions

Column's encoded vector *<encoded_value, CSN>*
<10, 250>
<301, 350>
...

Column's Dictionary	
index (implicit)	**dictionary value**
10	“Hello” <actual data of rid = 1>
20	“How are you” <updated actual data of rid = 1>
…	…
300	“Zebra” <the last item>

Delta space	
index (implicit)	**Value**
301	“Hello1” <updated actual data of rid = 1>
…	

Data structure in C++:

```
struct version_column {

        unsigned long encoded_value;        // = position in encoding dictionary

        unsigned long csn;

        version_column* next;        // pointer to previous version

}

vector<version_column*> column_versionSpace;  //vector of version pointer

vector<string> dictionary;                // the same dictionary with Data space

vector<string> delta_space;
```

- **Hash table**: Each entry consists of a row id in Data space and a pointer to the newest version of this row id in Version space. Pointer is actual the index of the version on Version space vector.

Structure of Hash table:

Row Id	Index
1 <rid = 1>	3 = Index of this version on Version space vector

Data structure in C++:

map<unsigned long, int> hashtable;

2. Version creating, filtering and garbage collection

2.1. Version creating:

- When a row data is updated by a Transaction then we will **first create a new version** of this row data **on Version space** then use the **Garbage collection** to **period update the data on Data space**.

- **First**: We will **look up the updated value in Column's Dictionary** to find its **position**. If **not found** then the updated data will be **added to Delta space** and return the **position from Delta space**. The Delta space also **maintain the order** as in Dictionary.

+ Old value = "Hello" => index = 10 on Dictionary

+ Updated value = "How are you" => found in position with index = 20 on Dictionary => no update Delta space, return encoded value = 20 to new Version.

+ Updated value = "Zoo" => not found on Dictionary => update Delta space, return position 301 from Delta space for encoded value of new Version.

- A **new object of struct version_column** is created with encoded_value = the position return from Dictionary or Delta space, CSN = Transaction's Commit Time stamp, pointer = NULL.

version_column { encoded_value = 20, csn = 350, next = NULL };

- Then we will check in **Hash table** to find the **previous version** of this row data. If **found an entry on Hash table** with rid = this row data's row id then we get **pointer to previous version** on Version space. We **point the next pointer of new created version to previous version** and **replace the previous version on Version space vector by new version** => the index on Hash table will not change.

- If we **do not find any entry on Hash table** with rid = this row data's row id then we will **add new version to Version space vector** and **create a new entry on Hash table as <rid, index on Version space vector>**.

- Later, the Garbage collection will update Data space to newest version from Version space. We will describe detail later.

2.2. Version filtering:

- Given a transaction with Start Time stamp **ts** wants to select a data row with row id = **rid**.

- **First**: we will find **an entry on Column's encoded value at index = rid**. If this entry has version_flag = 1 then we will check continue on Hash table otherwise we will compare Transaction's Start Time stamp ts with this entry's CSN. If **ts >= CSN** then this entry data is in **Transaction's result**, otherwise it is not in result because of the Transaction's Start time is earlier than the Commit Time stamp of the data. After finding the entry then we will **look up on Dictionary** to find the actual data of this row's column.

- If **version_flag = 1**, we will check on the **Hash table** to find **an entry with this rid**. We will find the **index of the latest version on Version space**.

- We will **traverse all versions** on Version space **from newest to oldest by using next pointer** to find appropriate version with **CSN <= ts**. If no version found then there are not versions value for this Transaction.

2.3 Garbage Collecting:

- The Garbage collection will maintain a list of rid that recently updated: vector<unsigned long> recently_updated_rids;

- The Garbage collection will run through the Data space and Version space with each rid that recently updated. It will **copy the encoded data of latest Version with latest CSN** to Column's encoded vector in Data space and update CSN of the entry equals to latest CSN.

- The Garbage collection will **remove all old Versions that are not read by any Transactions** (by checking in **Transaction queue**). It will remove by traverse by next pointer through all versions and delete old version not used anymore and **update the next pointer to NULL**. The **Hash table does not need to update** because we will not remove the latest version.

3. Write conflict:

- The write conflict **happens when 2 Transactions write to 1 data column at the same time**. As a result, we need a method to **lock data column** when a Transaction has started to write on it so that another transaction cannot write again.

- When a Transaction wants to write on a data with **rid**, it first compares its Start Time stamp with the CSN of the column entry on Data space Column's encoded vector with **index = rid**. If Transaction's **Start Timestamp >= CSN** then there are no other Transactions writing on this data -> it can write.

- So this Transaction will update the column entry’s **CSN field to MAX_VALUE** of its data type (for example MAX_VALUE of long data type, ~ 2^31 -1), so that other Transactions cannot write to this rid (because other Transaction’s Start Time stamp will **always LESS THAN CSN** value of this column's rid).

- After the Transaction **adds new version on Version space** for this column's rid and **commits** then **the Garbage collection** will update this column entry’s **CSN field to Transaction’s Commit Timestamp** and other Transaction can update again.

- Other Transactions which cannot update on the column rid because of writing conflict (implemented above) will be added to a **Queue** and then will be **restarted after some waiting time** to a new Start Time stamp. If the **new Start Time stamp >= column rid’s CSN** then it can write to the rid.